Robot Navigation by Conditional Sequencing

(Submitted to the 1994 IEEE Conference on Robotics and Automation)

Erann Gat
California Institute of Technology
Department of Mechanical Engineering
Jet Propulsion Laboratory
gat@robotics.jpl.nasa.gov

Greg Dorais
University of Michigan
Department of Computer Science

ABSTRACT

We investigate the application of conditional sequencing to robot navigation. Initial experimental results indicate that very robust navigation can bc achieved by layering a conditional sequencer on top of a set of simple sensorimotor behaviors. The approach is uniquely flexible, permitting very complex tasks to bc programmed reliably in very short periods of time. The tcchniquc was used in the recent American Association of Artificial Intelligence mobile robot contest. All of the contest-specific code was written in three days by a single programmer. The robot turned in the best overall performance of any entry. In addition, wc present the results of over ninety formal experimental runs performed under a variety of circumstances.

1. Introduction

In earlier work wc introduced a technique for mobile robot navigation based on conditional sequencing [Gat91]. At that time wc presented anecdotal evidence to support the claim that conditional sequencing is an effective technique for mobile robot navigation. In this paper wc describe ongoing experimental studies of autonomous navigation based on conditional sequencing.

Conditional sequencing (also called reactive plan execution) is an Artificial Intelligence software technology for controlling autonomous systems in unpredictable environments [Firby89, McDermott93]. To dale, conditional sequencing systems have shown great promise controlling systems in complex simulated environments. However, the technique has not been extensively applied to real robots. In this paper wc present some initial investigations in this direction, as well as some preliminary but very promising experimental results. To date wc have performed over ninety formal experimental runs.

Informally, conditional sequencing is a technique for getting a robot to follow instructions. Conditional scqucnccs arc distinguished from the more classical notion of a plan in that the instructions are not necessarily linear scqucnccs of steps. Instead, the execution of conditional scqucncc is dependent on the situation that unfolds during execution, Conditional sequences may include constraints on situations which do not manifest themselves until run lime.

The following is an informal example of a conditional sequence:

> To assemble a widget, locate Part A and Part B and a l/2-inch screw. Insert Part A into the hole in Part R and fasten with the screw. Take care not to over tighten the screw. If Part A dots not slide into the hole easily, usc a piece of sandpaper to remove any protruding bumps.

This is a set of instructions, but it is not a linear sequence. For example, the instruction "take care not 10 over lighten the screw" is not a step 10 bc performed *after* the prcccding onc ("fasten with the screw"); it is a *constraint* to bc applied simultaneously with the fastening operation. The last instruction is an example of a *contingency procedure.* It is normally not performed at all, but only in the case where something goes wrong,

Conditional scqucnccs work by invoking lower-level sequences. For example, locating Parts A and B involves invoking a procedure for searching for an object, which carries with it its own constraints about where to search, and contingency procedures about what to do if the search fails (look underneath things, for example). Robustness is

achieved by having a large number of contingency procedures which can recover from failures. Thus, the system can bc made reliable even though failures arc common provided that the failures arc made cognizantly, that is, provided that the robot can detect the failures.

Conditional sequencing has been applied mostly to high-level task execution. In this paper wc investigate the applicability of the technique to mobile robot navigation, The intuition behind this approach is that robots can navigate by following instructions similar to those that humans give to guide others to unknown destinations. For example:

> To get to the lab, go out the door, follow the hallway to the left. Turn right at the second corridor, Go around the big planter in the middle of the hall and turn at the third door on the right. If the hall is blocked, go around the other way, or go back to the office and ask for help,

This top-level sequence would bc expanded in terms of procedures for following halls, turning around corners, moving around obstacles, etc. The hierarchy bottoms out in primitive procedures which arc simply sensorimotor control laws that arc engineered by hand.

Wc advocate bottom-up development, where 10W-1CVC1 primitives arc designed first and their performance is empirically characterized and debugged, By performing empirical verification of primitives wc avoid the extremely difficult problem of analyzing the interactions of the robot with its environment.

Sequences arc then built on top of the debugged primitives, This is similar to the development methodology advocated by Brooks in his subsumption architecture [Brooks86] in which complex behaviors arc built on top of simpler ones, but it differs in the manner in which the layers interact. In subsumption the layers interact by arbitration: higher layers override lower ones and seize control of the vehicle's actuators. In our approach, higher layers provide input to the lower layers in order to parameterize their actions. Only the lowest layer actually controls the robot's actuators directly. More details on our architecture and development methodology can bc found in [Gat92]. There arc a number of researchers pursuing similar approaches including Connell [Conncl191], Arkin [Arkin90], and Bonnasso [Bonnasso92]. It is interesting to compare our work with Simmons [Simmons90] who advocates top-down development rather than bottom-up.

2. The Robot

21 Hardware

Our experiments were performed with 100% off-the-shelf commercially available hardware. This presents a unique opportunity to perform independent verification of our experimental results. Our software can bc used without modification by any researcher with the following hardware. (In fact, wc used a pair of identical robots during the course of the experiments, both using the same software.)

Each robot used in our experiments is a Real World Interface (RWI) B12 base with an 8-inch development enclosure. The B 12 is a 12-inch diameter circular base with a three-wheel synchrodrive mobility mechanism. The robot is thus capable of turning in place, and can travel along any path (although paths with discontinuous curvature require the robot to stop), The circular shape, synchrodrive, and the B 12's robust design make it a very convenient research platform.

The development enclosure, also from RWI, houses a Gespak MPL-4080 68000-based single-board computer, a ring of twelve Polaroid ultrasonic sensors, and a controller board for the sonars. On top of the development enclosure sits an Apple Macintosh Powcrbook. The connection between the Macintosh and the rest of the robot is an RS-232 link, so a desktop Mac could also bc used with an appropriate tether or wireless serial link.

Wc made onc modification to our robot. Because the robot has twelve sonars, each of which covers a 30-degree cone, there arc two different sonar configurations to choose from. (See figure 1.) The robot comes out of the box with a configuration shown in figure la. Wc changed this to the configuration shown in figure lb. Wc found that having a sonar pointing at the major compass points has some practical advantages when developing wall-following and obstacle-avoidance algorithms. The development enclosure is specifically designed to allow this reconfiguration, It takes about five minutes and requires no tools.

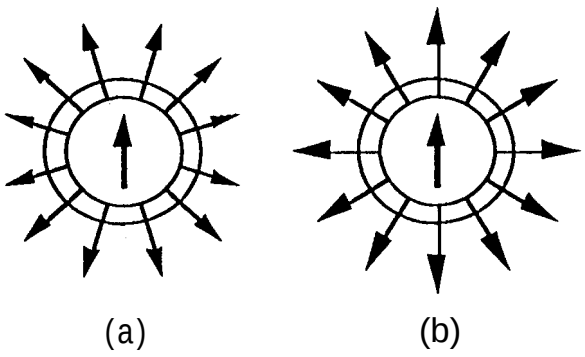

Figure 1: Two alternative sonar configurations. The central arrow shows the robot's direction of travel.

2.2 software

All development was done using Macintosh Common Lisp version 2 running under system 7.1. Embedded in the Lisp environment is a custom development system for the 68000 processor on the robot. This system includes an assembler, an operating system for the 68000, and a compiler for the ALFA programming language [Gat91] which is used to program primitive behaviors,

In the experiments dcscribcd in this paper conditional sequencing was done using two different systems. The first experiment (described in section 3) used a simplified conditional sequencer implemented directly in Lisp using the Lisp catch/throw construct. The second experiment (section 4) was implemented using Firby's Reactive Action Package (RAP) system [Firby89]. The RAP system has recently been re-implemented in Common Lisp, and is freely available for non-commercial USC. Wc modified Firby's system slightly by adding an interface to the robot. Both of these systems will be described in more detail in the appropriate section.

2.3 Primitive Behaviors

Our development methodology begins with the design of simple sensorimotor behaviors which arc used a building blocks for more complex activities. In this case wc arc concerned with navigation in an indoor environment where walls and doorways can be used as positional cues. The robot must be able to deal with unexpected obstacles, but must not rely too heavily on dead reckoning. Because of rotational drift, dead reckoning errors increase faster than the square of the distance traveled. Dead reckoning is reliable only for distances of about five **meters or** less.

For indoor environments wc usc three basic behaviors: dead-reckoning to a position while avoiding obstacles, following walls, and aligning to walls. This repertoire is generic to many tasks, and has not changed since wc began this work three years ago, although wc have fine-tuned the algorithms a bit since then.

The dead-reckoning behavior works as follows. The robot turns towards the designated goal location and begins to move forward. If the robot detects an obstacle some distance away it slows down and veers away from the obstacle. If the robot detects an obstacle nearby it stops and turns in onc direction (chosen according to the location of the obstacle and the goal) until it is able to move forward by a small amount. This "directional latching" prevents the robot from oscillating back and forth in a never-ending loop. The threshold distances for these two behaviors arc determined empirically.

Wall-following is performed by simply servoing to onc of the side-looking sonars. The algorithm contains a number of hacks to keep the robot stable and to handle doors. The wall-following algorithm also stop for obstacles. (Reliably moving around obstacles while following a wall is an unsolved problem.) There is no theoretical justification for the algorithm. In fact, it can be shown mathematically that the naive algorithm ought to be unstable. Nevertheless, the algorithm dots work in practice with high reliability.

Wall alignment is done by servoing the robot's heading until two adjacent sonars read the same distance, Experimentally this simple algorithm aligns the robot to smooth surfaces with an accuracy of better than onc degree. Wall alignment is used to correct for rotational drift brought about by slight misalignments in the robot's wheels, The result is dramatically improved dead-reckoning performance.

2.4 Cognizant Failure

There arc certainly better algorithms for performing all of these functions than the ones just described. However, the point is not **10** develop bullet-proof algorithms, but rather to show that conditional sequencing works. In fact, to test conditional sequencing it is better that the primitive behaviors not work perfectly because we wish to demonstrate that conditional sequences can recover

from failures, and thus produce reliable performance even when the primitives arc unreliable.

In order to recover from failures it is necessary for the robot to bc able 10 detect failures when they occur. Failures which the robot can detect wc term *cognizant failures*. To produce cognizant failure wc augment the primitive behaviors with monitoring routines which check to see that things arc working properl y. These monitor routines vary from situation to situation, but arc all fairly straightforward.

The dead-reckoning monitor checks the robot's position to scc if it is outside of a bounding region. It also imposes a time limit on reaching the destination, The time limit and the size of the bounding region arc parameters which can bc changed by the sequencer. Default values arc computed according to the initial distance to the goal,

The wall-following monitor checks the robot's heading to make sure that it is not turning faster than expected duc to rotational drift. It also makes sure that the distance to the wall is within an expected range, and that the robot is not stopped duc to an obstacle.

The wall-alignment monitor checks that the heading correction produced by wall alignment is not greater than an expected threshold. It also imposes a time limit on the al ignment process.

The sequencing infrastructure allows ncw monitors to bc defined by the programmer if desired.

2.5 Basic Sequences

As an illustration of how conditional sequencing works wc describe a fcw basic low-level sequences which wc used in a number of our experiments.

The dead-reckoning and wall-alignment primitives can bc used to construct more robust dead-reckoning and wall-a ignment sequences simply by retrying the primitive if it fails. For an example of how this works consider the situation in figure 2. The robot has to dead-reckon to a location which is blocked by an obstacle. Since the robot has no global map, and there arc no local indications of which direction is preferable for circumnavigating the obstacle, the robot chooses a direction at random, Suppose it dccidcs to go to the right, which happens to lead to an inefficient path. At some point the robot monitor indicates a cognizant failure when the robot moves outside the bounds of the monitor region. At this point, the sequencer restarts the primitive. The robot turns to face the goal, at which point the obstacle is on the robot's right, The robot therefore veers away to its left, and eventually moves around the obstacle in the correct direction.

Of course, there are situations where this strategy fails. However, such high-level failures arc also cognizant failures, and thus can be dealt with by higher-level recover procedures.

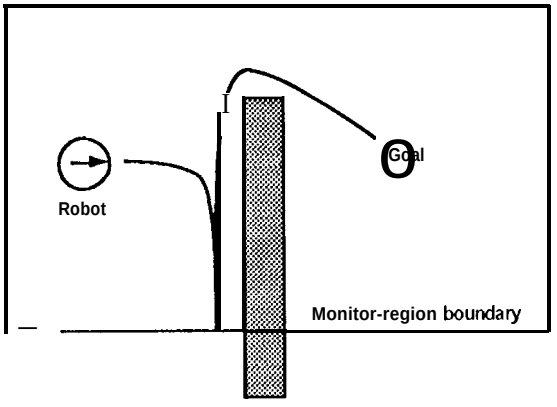

Figure 2: A basic sequence.

A second example of a basic sequence is following a wall past open doors. Normally an open door causes the wall-following primitive to fail cognizantly, since there is no longer a wall to follow. If the robot needs to move past open doors it simply invokes the dead-reckoning primitive to move past the door, and then starts following the wall again. This sequence requires its own monitor to make sure that it manages to find the wall on the other side of the door. If it dots not, the robot must first realign itself to the wall before proceeding.

By building up successively more complex repertoires of conditional sequences wc have been able to achieve very complex goal-directed behavior, as described in the following sections.

3. Experiment 1: The AAAI Robot Contest

The hardware and computational infrastructure described above was used to program an entry for the second American Association of Artificial Intelligence (AAAI) mobile robot contest. All of the contest-specific code was written in three days by a single programmer. The robot turned in the best overall performance of any entry in the events entered.

The contest consisted of three events, of which only the first two were entered. (The third event involved manipulating large cardboard boxes which our small robot was physically incapable of doing.) The contest took place in a large arena made of reconfigurable panels which were rearranged for the various events.

3.1 Sequencing Infrastructure

The sequencing infrastructure used in these experiments was a simple conditional sequence compiler implemented directly in Common Lisp using the catch/throw facility of that language [Steele90].

For this simplified sequencer wc made the assumption that scquencc execution was nominally linear except when a cognizant failure occurred. Wc therefore implemented as our basic sequencing construct a macro called with- recovery-procedures which has the following syntax:

```
(with-recovery-procedures
 timeout action
  (failure-code recovery-
 procedure )
  (failure-code recovery-
 procedure)
  ...)
```

Monitor procedures signaled a failure by calling a function called f a i l which took as an argument a Failure code which indicated the type of failure. This failure code was propagated upwards through nested with-recovery-procedure forms until onc was found which had a recovery proccdurc for that failure code. The top-level interpreter had a global recovery procedure for all failure codes which turned all the robot's motors off.

On lop of this basic sequencing facility wc implemented a number of additional facilities, including linear and non-linear sequences, state machines, and a macro for retrying a sequence a variable number of limes.

3.2 Event 1: Escape from the Office

The first event required the robot 10 explore a mock-up office and find its way out. The "office" was a 4-by-6-meter space containing real office furniture (but no carpeting), including a desk, some chairs, a filing cabinet and a cabinet-bookshelf. There were three "doors" in the office, consisting of movable panels, Onc minute into the contest onc of the three doors, chosen at random by the judges, was opened. The robot had to find the open door, go through it, and then navigate an obstacle field to a finish line some fifteen meters distant. The robot had to stop within two meters of the finish line. The robot was starred at a known orientation but at an unknown randomly chosen location within the Oft-lee.

The strategy used was the following. The robot wandered randomly around the office for onc minute (the time that the doors were guaranteed to bc closed) while keeping track of its extremes of x-y positions, Using this data it computed x-y positions that would guarantee that it was outside of the office. It then began to try each door in turn until it reached a location outside of the office. It then moved to the finish line using the dead-rcckoning primitive.

The robot was tested about a dozen times during the preparation for the contest. It never failed, During the contest the judges consistently chose to open the door that happened to result in the worst-case performance. Nevertheless, the robot placed second overall in this event.

On a whim, wc also implemented procedures for entering and exiting the arena autonomously. Because all of the sequencing infrastructure was already developed all wc had to do was write a single top-level sequence. This took about ten minutes to do. It was tested six times (including the two official contest runs) and worked every time.

3.3 Event 2: Deliver the Coffeepot

The second event was much more complex. The event took place in a 15-by-24-meter area which had been partitioned into a maze of offices and hallways, (SW figure 3.) There was no furniture this time; instead cardboard boxes were placed in the arena to serve as obstacles. The robots were allowed to have *a priori* information about the layout of the offices and hallways, but not the locations of the boxes (which could bc changed from run to run),

The objective in this contest was to find a coffeepot which was located somewhere in the arena and deliver it to a designated location, To accommodate robots without any manipulation hardware the robots were not required to actually pick up the coffeepot; it was sufficient for the robot to move near the coffeepot and indicate that it knew that it had located the pot. The robot was told ahead of time which quadrant the coffeepot was in, and the location of the destination.

The robot was not told its initial location nor it initial orientation, The robot therefore had to

begin by self-localizing itself to determine its location in the map. This was done as follows. The robot began by searching for a wall (in a manner described below). Once it found onc it began following the wall while making a record of the pattern of left and right turns it was making. It turns out that this pattern was unique for each of the wall assemblies in the arena. The robot was thus able to determine its position after locating and circumnavigating a wall assembly.

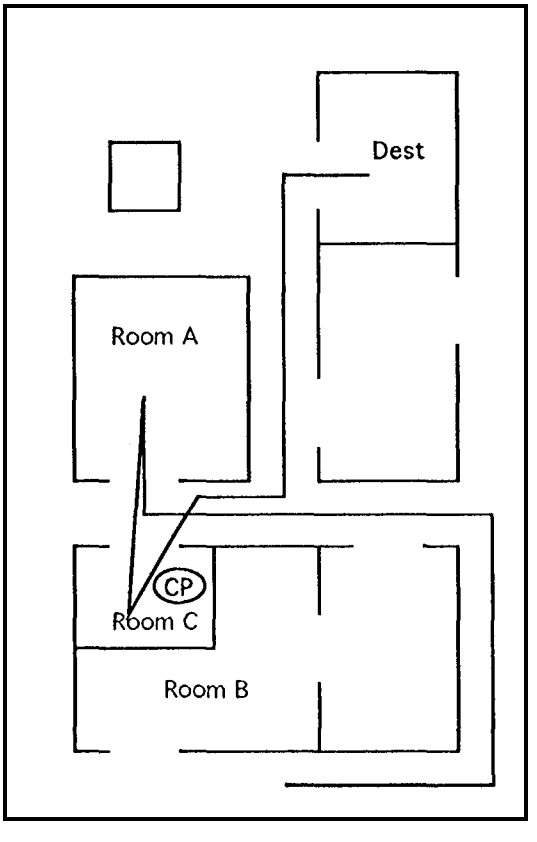

Figure 3: The layout for experiment 2.

Once the robot had self-localized the rest of the event was straightforward. The robot used a simple path planner to plan a sequence of rooms to visit to search for the coffeepot, which was located in room C. Because the robot knew the quadrant where the coffeepot was located it only had to search three rooms, It found the coffeepot in the second room, (Actually, since the robot had no sensors capable of detecting the coffeepot it had to bc told that it was in the same room.) The robot then moved directly to the delivery destination. The robot's general route after self-localizing is shown in figure 3. This depiction leaves out the avoidance maneuvers the robot performed to avoid boxes. These were not recorded at the time.

This experiment was only performed once. However, our robot was the only robot in the contest to complete this event. It also took a single programmer lcss than eighteen hours 10 program the robot for this event.

4. Experiment 2: Trek to the Lab

The preceding results, while encouraging, represent only anecdotal data, and is thus of limited utility for drawing general conclusions about conditional sequencing. In order to provide some more rigorous data wc set out to perform an experiment with a statistically significant 'number of trials,

The environment wc used was the hallways of the building where wc work, a typical modern office building consisting of a maze of orthogonal hallways with doorways and random obstacles. We wanted to show that the performance of the robot improved in a predictable, monotonic manner as recovery procedures were added to sequences as they were being tested.

For these experiments wc used a different conditional sequencing infrastructure. Instead of the simplified systcm used in the previous experiments wc used a slightly modified version of Firby's Reactive Action Package (RAP) systcm. RAPs arc a sophisticated conditional sequencing language whose details arc beyond the scope of this paper. The system is dcscribcd in detail in [Firby89]. To date, RAPs have been used to produce very sophisticated behavior in simulated environments, but have not been used on real robots, These experiments bridge the gap between a sophisticated AI reactive execution system and robot hardware (cf. [Georgeff87]).

Wc chose as our initial benchmark task to navigate from a particular office to the mail room which was approximately ten meters away on the opposite side of the hallway. This turned out not to bc a sufficiently difficult task. Wc performed thirty trials with an initial version of the software; all but onc worked perfectly.

As this left little room for improvement, wc chose a more difficult task. This time wc programmed the robot to navigate to our lab which was halfway across the building, a distance of approximately seventy meters involving about a dozen turns.

We defined a success metric based on the distance traveled before an unrecoverable failure, and ran fifteen trials, adding recovery procedures as problems occurred. By the tenth trial the robot was consistently completing lhc run, (See figure 4.)

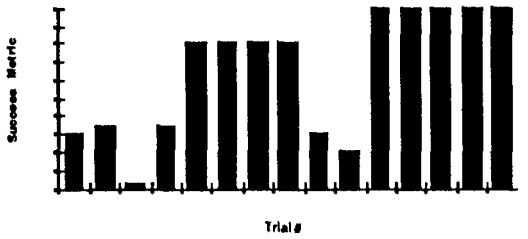

Figure 4: Fifteen consecutive runs from the office to the lab.

Wc once again raised our sights and programmed the robot to return from the lab to the office, The total length of the run was now nearly 150 meters, Wc ran forly-seven experiments, again making adding recovery procedures as problems occurred. The results are shown in figure 5. The results arc not as consistent as lhc previous experiment, but a clear upward trend in performance can be seen.

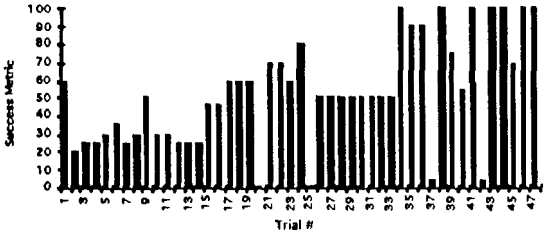

Figure 5: Forty-seven consecutive runs from the office to the lab and back,

The total development time represented by all ninety-two of these experiments (including the thirty runs to the copier room) was approximate] y three weeks. Of course, this dots not include the time spent developing the conditional sequencing infrastructure and primitive behaviors.

S. Conclusions and Future Work

Wc have presented further experimental evidence of the efficacy of conditional sequencing for controlling real-world autonomous mobile robots in indoor environments. Wc described the results of over ninety experimental runs in a variety of situations using two different conditional sequencing infrastructures. In all cases, robust effective performance was achieved with very little programming effort once the infrastructure was in place.

Wc consider these results preliminary despite the large number of trials relative to other published studied of this kind. Wc arc currently working on a design for a rigorous experimental protocol for comparing different conditional sequencing approaches and other navigation algorithms.

Wc arc also working on extending this work in two different directions. First, wc plan to investigate the application of conditional sequencing to outdoor navigation, Second, wc arc investigating compilation techniques to allow conditional sequencing to run on small processors such as those available for planetary *rovers.* Some preliminary work in this direction has already been done [Gat93].

Acknowledgements

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

References

[Arkin90] Ronald C. Arkin, "Integrating Behavioral, Perceptual and World Knowledge in Reactive Navigation," *Robotics and Autonomous Systems,* vol. *6,* pp. 105-122, 1990.

[Bonasso92] R. Peter Bonasso, "Using Parallel Program Specifications For Reactive Control of Underwater Vehicles," *Journal of Applied Intelligence,* Kluwer Academic Publishers, Norwell , MA, June 1992.

[Brooks86] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal on Robotics and Automation,* vol RA-2, no. 1, March 1986.

[Connell91] Jonathan Connell, "SSS: A Hybrid Architecture Applied to Robot Navigation," unpublished manuscript,

[Firby89] R. James Firby, *Adaptive Execution in Dynamic Domains,* Ph.D. thesis, Yale University Department of Computer Science, 1989.

[Gat91b] Erann Gat, "ALFA: A Language for Programming Reactive Robotic Control Systems",*IEEE Conference on Robotics and Automation,* 1991,

[Gat91d] Erann Gat, "Low-compilation Sensor-driven Control for Task-directed Navigation," *IEEE Conference on Robotics and Automation,)991.*

[Gat92] Erann Gat, "Integrating Reaction and Planning in a Heterogeneous Asynchronous Architecture for Controlling Real World Mobile Robots," *Proceedings of the Tenth National Conference on Artificial Intelligence* (AAAI), 1992.

[Gat93] Erann Gat, et al., "Behavior Control for Planetary Exploration," *IEEE Transactions on Robotics and Automation,* to appear.

[Georgeff87] Michael Georgeff and Amy Lanskey, "Reactive Reasoning and Planning", *Proceedings of AAAI-87.*

[McDermott91] Drew McDermott, "A Reactive Plan Language," Technical Report 864, Yale University Department of Computer Science.

[Simmons90] Reid Simmons, "An Architecture for Coordinating Planning, Sensing and Action," *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, 1990.*

[Steele90] Guy Steele, *Common Lisp: The Language* (second edition), Digital Press, 1990.